Project -- Process Scheduling

Process States

This program, process-run.py, allows you to see how process states change as programs run and either use the CPU (e.g., perform an add instruction) or do I/O (e.g., send a request to a disk and wait for it to complete). See the README for details.

1. Run **process-run.py** with the following flags: **-l 5:100,5:100**. What should the CPU utilization be (e.g., the percent of time the CPU is in use?) Why do you know this? Use the **-c** and **-p** flags to see if you were right.

python ./process-run.py -l 5:100,5:100

```
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -l 5:100,5:100 -c -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1      RUN:cpu       READY            1
  2      RUN:cpu       READY            1
  3      RUN:cpu       READY            1
  4      RUN:cpu       READY            1
  5      RUN:cpu       READY            1
  6         DONE     RUN:cpu            1
  7         DONE     RUN:cpu            1
  8         DONE     RUN:cpu            1
  9         DONE     RUN:cpu            1
 10         DONE     RUN:cpu            1

Stats: Total Time 10
Stats: CPU Busy 10 (100.00%)
Stats: IO Busy  0 (0.00%)
```

2.Now run with these flags: **./process-run.py -l 4:100,1:0**. These flags specify one process with 4 instructions (all to use the CPU), and one that simply issues an I/O and waits for it to be done. How long does it take to complete both processes? Use **-c** and **-p** to find out if you were right.

python ./process-run.py -l 4:100,1:0

```
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu

Process 1
  io

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -l 4:100,1:0 -c -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1      RUN:cpu       READY            1
  2      RUN:cpu       READY            1
  3      RUN:cpu       READY            1
  4      RUN:cpu       READY            1
  5         DONE      RUN:io            1
  6         DONE     WAITING                       1
  7         DONE     WAITING                       1
  8         DONE     WAITING                       1
  9         DONE     WAITING                       1
 10*        DONE        DONE

Stats: Total Time 10
Stats: CPU Busy 5 (50.00%)
Stats: IO Busy  4 (40.00%)
```

3.Switch the order of the processes: **-l 1:0,4:100**. What happens now? Does switching the order matter? Why? (As always, use -c and -p to see if you were right)

python ./process-run.py -l 1:0,4:100

```
Produce a trace of what would happen when you run these processes:
Process 0
  io

Process 1
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -l 1:0,4:100 -c -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING     RUN:cpu            1            1
  3      WAITING     RUN:cpu            1            1
  4      WAITING     RUN:cpu            1            1
  5      WAITING     RUN:cpu            1            1
  6*        DONE        DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
```

4.We'll now explore some of the other flags. One important flag is -S, which determines how the system reacts when a process issues an I/O. With the flag set to SWITCH ON END, the system will NOT switch to another process while one is doing I/O, instead waiting until the process is completely finished. What happens when you run the following two processes (**-l 1:0,4:100 -c -S SWITCH_ON_END**), one doing I/O and the other doing CPU work?

python ./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_END

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING      READY                         1
  3      WAITING      READY                         1
  4      WAITING      READY                         1
  5      WAITING      READY                         1
  6*        DONE     RUN:cpu            1
  7         DONE     RUN:cpu            1
  8         DONE     RUN:cpu            1
  9         DONE     RUN:cpu            1
```

python ./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_END -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING       READY                         1
  3      WAITING       READY                         1
  4      WAITING       READY                         1
  5      WAITING       READY                         1
  6*        DONE     RUN:cpu            1
  7         DONE     RUN:cpu            1
  8         DONE     RUN:cpu            1
  9         DONE     RUN:cpu            1

Stats: Total Time 9
Stats: CPU Busy 5 (55.56%)
Stats: IO Busy  4 (44.44%)
```

5.Now, run the same processes, but with the switching behavior set to switch to another process whenever one is **WAITING** for I/O (**-l 1:0,4:100 -c -S SWITCH ON IO**). What happens now? Use **-c** and **-p** to confirm that you are right.

./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING     RUN:cpu            1            1
  3      WAITING     RUN:cpu            1            1
  4      WAITING     RUN:cpu            1            1
  5      WAITING     RUN:cpu            1            1
  6*        DONE        DONE
```

./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING     RUN:cpu            1            1
  3      WAITING     RUN:cpu            1            1
  4      WAITING     RUN:cpu            1            1
  5      WAITING     RUN:cpu            1            1
  6*        DONE        DONE

Stats: Total Time 6
Stats: CPU Busy 5 (83.33%)
Stats: IO Busy  4 (66.67%)
```

6.One other important behavior is what to do when an I/O completes. With -I IO RUN LATER, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (Run **./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH ON IO -I IO RUN LATER -c -p**) Are system resources being effectively utilized?

python ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER

```
Produce a trace of what would happen when you run these processes:
Process 0
  io
  io
  io

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Process 2
  cpu
  cpu
  cpu
  cpu
  cpu

Process 3
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_LATER -c -p

```
Time      PID: 0     PID: 1     PID: 2     PID: 3        CPU        IOs
  1       RUN:io      READY      READY      READY          1
  2      WAITING    RUN:cpu      READY      READY          1          1
  3      WAITING    RUN:cpu      READY      READY          1          1
  4      WAITING    RUN:cpu      READY      READY          1          1
  5      WAITING    RUN:cpu      READY      READY          1          1
  6*       READY    RUN:cpu      READY      READY          1
  7        READY       DONE    RUN:cpu      READY          1
  8        READY       DONE    RUN:cpu      READY          1
  9        READY       DONE    RUN:cpu      READY          1
 10        READY       DONE    RUN:cpu      READY          1
 11        READY       DONE    RUN:cpu      READY          1
 12        READY       DONE       DONE    RUN:cpu          1
 13        READY       DONE       DONE    RUN:cpu          1
 14        READY       DONE       DONE    RUN:cpu          1
 15        READY       DONE       DONE    RUN:cpu          1
 16        READY       DONE       DONE    RUN:cpu          1
 17       RUN:io       DONE       DONE       DONE          1
 18      WAITING       DONE       DONE       DONE                     1
 19      WAITING       DONE       DONE       DONE                     1
 20      WAITING       DONE       DONE       DONE                     1
 21      WAITING       DONE       DONE       DONE                     1
 22*      RUN:io       DONE       DONE       DONE          1
 23      WAITING       DONE       DONE       DONE                     1
 24      WAITING       DONE       DONE       DONE                     1
 25      WAITING       DONE       DONE       DONE                     1
 26      WAITING       DONE       DONE       DONE                     1
 27*        DONE       DONE       DONE       DONE

Stats: Total Time 27
Stats: CPU Busy 18 (66.67%)
Stats: IO Busy  12 (44.44%)
```

7.Now run the same processes, but with **-I IO RUN IMMEDIATE** set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

python ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_IMMEDIATE

```
Produce a trace of what would happen when you run these processes:
Process 0
  io
  io
  io

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Process 2
  cpu
  cpu
  cpu
  cpu
  cpu

Process 3
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run IMMEDIATELY
```

python ./process-run.py -l 3:0,5:100,5:100,5:100 -S SWITCH_ON_IO -I IO_RUN_IMMEDIATE -c -p

```
Time     PID: 0     PID: 1     PID: 2     PID: 3        CPU        IOs
  1      RUN:io      READY      READY      READY          1
  2     WAITING    RUN:cpu      READY      READY          1          1
  3     WAITING    RUN:cpu      READY      READY          1          1
  4     WAITING    RUN:cpu      READY      READY          1          1
  5     WAITING    RUN:cpu      READY      READY          1          1
  6*     RUN:io      READY      READY      READY          1
  7     WAITING    RUN:cpu      READY      READY          1          1
  8     WAITING       DONE    RUN:cpu      READY          1          1
  9     WAITING       DONE    RUN:cpu      READY          1          1
 10     WAITING       DONE    RUN:cpu      READY          1          1
 11*     RUN:io       DONE      READY      READY          1
 12     WAITING       DONE    RUN:cpu      READY          1          1
 13     WAITING       DONE    RUN:cpu      READY          1          1
 14     WAITING       DONE       DONE    RUN:cpu          1          1
 15     WAITING       DONE       DONE    RUN:cpu          1          1
 16*       DONE       DONE       DONE    RUN:cpu          1
 17        DONE       DONE       DONE    RUN:cpu          1
 18        DONE       DONE       DONE    RUN:cpu          1

Stats: Total Time 18
Stats: CPU Busy 18 (100.00%)
Stats: IO Busy  12 (66.67%)
```

8.Now run with some randomly generated processes: **-s 1 -l 3:50,3:50 or -s 2 -l 3:50,3:50 or -s 3 -l 3:50,3:50**. See if you can predict how the trace will turn out. What happens when you use the flag **-I IO_RUN_IMMEDIATE vs. -I IO_RUN_LATER**? What happens when you use **-S SWITCH_ON_IO vs. -S SWITCH_ON_END**?

python ./process-run.py -s 1 -l 3:50,3:50

```
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  io
  io

Process 1
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -s 1 -l 3:50,3:50 -c -p

Time	PID: 0	PID: 1	CPU	IOs
1	RUN:cpu	READY	1	
2	RUN:io	READY	1	
3	WAITING	RUN:cpu	1	1
4	WAITING	RUN:cpu	1	1
5	WAITING	RUN:cpu	1	1
6	WAITING	DONE		1
7*	RUN:io	DONE	1	
8	WAITING	DONE		1
9	WAITING	DONE		1
10	WAITING	DONE		1
11	WAITING	DONE		1
12*	DONE	DONE		

python ./process-run.py -s 2 -l 3:50,3:50

```
Produce a trace of what would happen when you run these processes:
Process 0
  io
  io
  cpu

Process 1
  cpu
  io
  io

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -s 2 -l 3:50,3:50 -c -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1       RUN:io       READY            1
  2      WAITING     RUN:cpu            1            1
  3      WAITING      RUN:io            1            1
  4      WAITING     WAITING                         2
  5      WAITING     WAITING                         2
  6*      RUN:io     WAITING            1            1
  7      WAITING     WAITING                         2
  8*     WAITING      RUN:io            1            1
  9      WAITING     WAITING                         2
 10      WAITING     WAITING                         2
 11*     RUN:cpu     WAITING            1            1
 12         DONE     WAITING                         1
 13*        DONE        DONE

Stats: Total Time 13
Stats: CPU Busy 6 (46.15%)
Stats: IO Busy  11 (84.62%)
```

python ./process-run.py -s 3 -l 3:50,3:50

```
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  io
  cpu

Process 1
  io
  io
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

python ./process-run.py -s 3 -l 3:50,3:50 -c -p

```
Time      PID: 0      PID: 1          CPU          IOs
  1      RUN:cpu       READY            1
  2       RUN:io       READY            1
  3      WAITING      RUN:io            1            1
  4      WAITING     WAITING                         2
  5      WAITING     WAITING                         2
  6      WAITING     WAITING                         2
  7*     RUN:cpu     WAITING            1            1
  8*        DONE      RUN:io            1
  9         DONE     WAITING                         1
 10         DONE     WAITING                         1
 11         DONE     WAITING                         1
 12         DONE     WAITING                         1
 13*        DONE     RUN:cpu           1

Stats: Total Time 13
Stats: CPU Busy 6 (46.15%)
Stats: IO Busy  9 (69.23%)
```

Process Schedule

This program, scheduler.py, allows you to see how different schedulers perform under scheduling metrics such as response time, turnaround time, and total wait time. See the README for details.

1.Compute the response time and turnaround time when running three jobs of length 200 with the SJF and FIFO schedulers.

python ./scheduler.py -p FIFO -l 200,200,200 -c

```
ARG policy FIFO
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

python ./scheduler.py -p SJF -l 200,200,200 -c

```
ARG policy SJF
ARG jlist 200,200,200

Here is the job list, with the run time of each job:
  Job 0 ( length = 200.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 200.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

2.Now do the same but with jobs of different lengths: **100, 200, and 300**.

python ./scheduler.py -p FIFO -l 100,200,300 -c

```
ARG policy FIFO
ARG jlist 100,200,300

Here is the job list, with the run time of each job:
  Job 0 ( length = 100.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 300.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 1 for 200.00 secs ( DONE at 300.00 )
  [ time 300 ] Run job 2 for 300.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 300.00  Wait 100.00
  Job   2 -- Response: 300.00  Turnaround 600.00  Wait 300.00

  Average -- Response: 133.33  Turnaround 333.33  Wait 133.33
```

python ./scheduler.py -p SJF -l 100,200,300 -c

```
ARG policy SJF
ARG jlist 100,200,300

Here is the job list, with the run time of each job:
  Job 0 ( length = 100.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 300.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 1 for 200.00 secs ( DONE at 300.00 )
  [ time 300 ] Run job 2 for 300.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 300.00  Wait 100.00
  Job   2 -- Response: 300.00  Turnaround 600.00  Wait 300.00

  Average -- Response: 133.33  Turnaround 333.33  Wait 133.33
```

3.Now do the same, but also with the **RR** scheduler and a time-slice of 1.

python ./scheduler.py -p RR -l 2,3,6 -q 1 -c

```
ARG policy RR
ARG jlist 2,3,6

Here is the job list, with the run time of each job:
  Job 0 ( length = 2.0 )
  Job 1 ( length = 3.0 )
  Job 2 ( length = 6.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job   0 for 1.00 secs
  [ time   1 ] Run job   1 for 1.00 secs
  [ time   2 ] Run job   2 for 1.00 secs
  [ time   3 ] Run job   0 for 1.00 secs ( DONE at 4.00 )
  [ time   4 ] Run job   1 for 1.00 secs
  [ time   5 ] Run job   2 for 1.00 secs
  [ time   6 ] Run job   1 for 1.00 secs ( DONE at 7.00 )
  [ time   7 ] Run job   2 for 1.00 secs
  [ time   8 ] Run job   2 for 1.00 secs
  [ time   9 ] Run job   2 for 1.00 secs
  [ time  10 ] Run job   2 for 1.00 secs ( DONE at 11.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 4.00  Wait 2.00
  Job   1 -- Response: 1.00  Turnaround 7.00  Wait 4.00
  Job   2 -- Response: 2.00  Turnaround 11.00  Wait 5.00

  Average -- Response: 1.00  Turnaround 7.33  Wait 3.67
```